Data Cleaning and Query Answering with Matching Dependencies and Matching Functions

Leopoldo Bertossi*
Carleton University
Ottawa, Canada
bertossi@scs.carleton.ca

Solmaz Kolahi
University of British Columbia
Vancouver, Canada
solmaz@cs.ubc.ca

Laks V. S. Lakshmanan
University of British Columbia
Vancouver, Canada
laks@cs.ubc.ca

ABSTRACT

Matching dependencies were recently introduced as declarative rules for data cleaning and entity resolution. Enforcing a matching dependency on a database instance identifies the values of some attributes for two tuples, provided that the values of some other attributes are sufficiently similar. Assuming the existence of matching functions for making two attributes values equal, we formally introduce the process of cleaning an instance using matching dependencies, as a chase-like procedure. We show that matching functions naturally introduce a lattice structure on attribute domains, and a partial order of semantic domination between instances. Using the latter, we define the semantics of clean query answering in terms of certain/possible answers as the greatest lower bound/least upper bound of all possible answers obtained from the clean instances. We show that clean query answering is intractable in some cases. Then we study queries that behave monotonically w.r.t. semantic domination order, and show that we can provide an under/over approximation for clean answers to monotone queries. Moreover, non-monotone positive queries can be relaxed into monotone queries.

Categories and Subject Descriptors

H.2.0 [**Database Management**]: General- Security, integrity, and protection; H.2.4 [**Database Management**]: Systems- Relational databases

General Terms

Algorithms, Theory

Keywords

Matching Dependency, Matching Function, Semantic Domination, Lattice, Certain Answer, Possible Answer

*Faculty Fellow of the IBM CAS. Also affiliated to University of Concepción (Chile).

ICDT 2011, March 21–23, 2011, Uppsala, Sweden.

1. INTRODUCTION

Matching dependencies (MDs) in relational databases were recently introduced in [17] as a means of codifying a domain expert's knowledge that is used in improving data quality. They specify that a pair of attribute values in two database tuples are to be matched, i.e., made equal, if similarities hold between other pairs of values in the same tuples. This is a generalization of entity resolution [16], where basically full tuples have to be merged or identified since they seem to refer to the same entity of the outside reality. This form of data fusion [11] is important in data quality assessment and in data cleaning.

Matching dependencies were formally studied in [18], as semantic constraints for data cleaning and were given a model-theoretic semantics. The main emphasis in that paper was on the problem of entailment of MDs and on the existence of a formal axiom system for that task.

MDs as presented in [18] do not specify *how* the matching of attribute values is to be done. In data cleaning, the user, on the basis of his or her experience and knowledge of the application domain, may have a particular methodology or heuristics for enforcing the identifications. *In this paper we investigate MDs in the context of matching functions*. These are functions that abstract the implementation of value identification. Rather than investigate specific matching functions, we explore a class of matching functions satisfying certain natural and intuitive axioms. With these axioms, matching functions impose a lattice-theoretic structure on attribute domains. Intuitively, given two input attribute values that need to be made equal, the matching function produces a value that *contains* the information present in the two inputs and *semantically dominates* them. We show that this semantic domination partial order can be naturally lifted to tuples of values as well as database instances as sets of tuples. The following example illustrates the role of matching functions.

EXAMPLE 1. Consider the following database instance D_0. Assume there is a matching dependency stating that if for two tuples the values of name and phone are similar, then the value of address should be made identical. Consider a similarity relation that indicates the values of name and phone are similar for the two tuples in this instance. To enforce the matching dependency, we create another instance D_1 in which the value of address for two tuples is the result of applying a matching function on the two previous addresses. This function combines the information in those address values, and thus D_1 semantically dominates D_0.

D_0	*name*	*phone*	*address*
	John Doe	(613)123 4567	Main St., Ottawa
	J. Doe	123 4567	25 Main St.

D_1	*name*	*phone*	*address*
	John Doe	(613)123 4567	25 Main St., Ottawa
	J. Doe	123 4567	25 Main St., Ottawa

We can continue this process in a chase-like manner if there are still other MD violations in D_1. ■

The framework of [18] leaves the implementation details of data cleaning process with MDs completely unspecified and implicitly leaves it to the application on hand. We point out some limitations of the proposal in [18] for purposes of cleaning dirty instances in the presence of multiple MDs. We also show that a formulation of the formal semantics of satisfaction and enforcement of MDs that incorporates matching functions remedies this problem. In giving such a formulation, we revisit the original semantics for MDs proposed in [18], propose some changes and investigate their consequences. More precisely, we define *intended clean instances*, those that are obtained through the application of the MDs in a chase-like procedure. We further investigate properties of this procedure in relation to the properties of the matching functions, and show that, in general, the chase procedure produces several different clean instances, each of which semantically dominates the original dirty instance.

We then address the problem of query answering over a dirty instance, where the MDs do *not* hold. We take advantage of the semantic domination order between instances, and define *clean answers* by specifying a tight lower bound (corresponding to certain answers) and a tight upper bound (corresponding to possible answers) for all answers that can be obtained from any of the possibly many clean instances. We show that computing the exact bounds is intractable in general. However, in polynomial time we can generate an under-approximation for certain answers as well as an over-approximation for possible answers for queries that behave *monotonically* w.r.t. the semantic domination order.

We argue that monotone queries provide more informative answers on instances that have been cleaned with MDs and matching functions. We therefore introduce new relational algebra operators that make use of the underlying lattice structure on the domain of attribute values. These operators can be used to *relax* a regular positive relational algebra query and make it monotone w.r.t. the semantic domination order.

Recently, Swoosh [8] has been proposed as generic framework for entity resolution. In entity resolution, whole tuples are identified, or merged into a new tuple, whenever similarities hold between the tuples on some attributes. Accordingly, the similarity and matching functions work at the tuple level. Given their similarity of purpose, it is interesting to seek the relationship between the frameworks of MDs and of Swoosh. We address this question in this paper.

In summary, we make the following contributions:

- We identify the limitations of the original proposal of MDs [18] w.r.t. the application of data cleaning in the presence of multiple MDs and show that the limitations can be overcome by considering MDs along with matching functions.
- We study matching functions in terms of their properties, captured in the form of certain intuitive and natural axioms. Matching functions induce a lattice framework on attribute domains which can be lifted to a partial order over instances, that we call semantic domination.
- We formally characterize answering a query given a dirty instance and a set of MDs, and capture it using certain and possible answers. Computing these answers is intractable in general. For queries that are monotone w.r.t. the semantic domination relation, we develop a polynomial time heuristic procedure for obtaining under- and over-approximations of query answers.
- We demonstrate the power of the framework of MDs and of our chase procedure for MD application by reconstructing the most common case for Swoosh, the so-called *union and merge* case, in terms of matching dependencies with matching functions.

The paper is organized as follows. In Section 2, we provide necessary background on matching dependencies as originally introduced. We introduce matching functions and the notion of semantic domination in Section 3. Then we define the data cleaning process with MDs in Section 4. We explore the semantic of query answering in Section 5. In Section 6, we study monotone queries and show how clean answers can be approximated. We establish a connection to an important related work, Swoosh, in Section 7, and present concluding remarks in Section 8. Some of the proofs can be found in [10].

2. BACKGROUND

A database schema $\mathcal{R}$ is a set $\{R_1, \ldots, R_n\}$ of relation names. Every relation R_i is associated with a set of attributes, written as $R_i(A_1, \ldots, A_m)$, where each attribute A_j has a domain Dom_{A_j}. We assume that attribute names are different across relations in the schema, but two attributes A_j, A_k can be *comparable*, i.e., $Dom_{A_j} = Dom_{A_k}$. An instance D of schema $\mathcal{R}$ assigns a finite set of tuples t^D to every relation R_i, where t^D can be seen as a function that maps every attribute A_j in R_i to a value in Dom_{A_j}. We write $t^D[A_j]$ to refer to this value. When X is a list of attributes, we may write $t^D[X]$ to refer to the corresponding list of attribute values. A tuple t^D for a relation name $R \in \mathcal{R}$ is called an R-tuple. We deal with queries $\mathcal{Q}$ that are expressed in relational algebra, and treat them as operators that map an instance D to an instance $\mathcal{Q}(D)$.

For every attribute A in the schema, we assume a binary similarity relation $\approx_A \subseteq Dom_A \times Dom_A$. Notice that whenever A and A' are comparable, the similarity relations $\approx_A$ and $\approx_{A'}$ are identical. We assume that each $\approx_A$ is symmetric and subsumes equality, i.e., $=_{Dom_A} \subseteq \approx_A$. When there is no confusion, we simply use $\approx$ for the similarity relation. In particular, for lists of pairwise comparable attributes, $X_i = A_1^i, \ldots, A_n^i$, $i = 1, 2$, we write $X_1 \approx X_2$ to mean $A_1^1 \approx_1 A_1^2 \wedge \cdots \wedge A_n^1 \approx_n A_n^2$, where $\approx_i$ is the similarity relation applicable to attributes A_i^1, A_i^2.

Given two pairs of pairwise comparable attribute lists X_1, X_2 and Y_1, Y_2 from relations R_1, R_2, resp., a *matching*

dependency (MD) [18] is a sentence of the form

$$\varphi\colon \quad R_1[X_1] \approx R_2[X_2] \rightarrow R_1[Y_1] \rightleftharpoons R_2[Y_2].^1 \qquad (1)$$

This dependency intuitively states that if for an R_1-tuple t_1 and an R_2-tuple t_2 in instance D, the attribute values in $t_1^D[X_1]$ are similar to attribute values in $t_2^D[X_2]$, then we need to make the values $t_1^D[Y_1]$ and $t_2^D[Y_2]$ pairwise identical.

Enforcing MDs may cause a database instance D to be changed to another instance D'. To keep track of every single change, we assume that every tuple in an instance has a unique identifier t, which could identify it in both instance D and its changed version D'. We use t^D and $t^{D'}$ to refer to a tuple and its changed version in D' that has resulted from enforcing an MD. For convenience, we may use the terms tuple and tuple identifier interchangeably.

Fan et al. [18] give a *dynamic semantics* for matching dependencies in terms of a pair of instances: one where the similarities hold, and a second where the specified identifications have been enforced:

Definition 1. [18] A pair of instances (D, D') satisfies the MD $\varphi : R_1[X_1] \approx R_2[X_2] \rightarrow R_1[Y_1] \rightleftharpoons R_2[Y_2]$, denoted $(D, D') \models \varphi$, if for every R_1-tuple t_1 and R_2-tuple t_2 in D that match the left-hand side of φ, i.e., $t_1^D[X_1] \approx t_2^D[X_2]$, the following holds in the instance D':

(a) $t_1^{D'}[Y_1] = t_2^{D'}[Y_2]$, i.e., the values of the right-hand side attributes of φ have been identified in D'; and

(b) t_1, t_2 in D' match the left-hand side of φ, that is, $t_1^{D'}[X_1] \approx t_2^{D'}[X_2]$.

For a set Σ of MDs, $(D, D') \models \Sigma$ iff $(D, D') \models \varphi$ for every $\varphi \in \Sigma$. An instance D' is called *stable* if $(D', D') \models \Sigma$. ■

Notice that a stable instance satisfies the MDs by itself, in the sense that all the required identifications are already enforced in it. Whenever we say that an instance is *dirty*, we mean that it is not stable w.r.t. the given set of MDs.

While this definition may be sufficient for the implication problem of MDs considered by Fan et al. [18], it does not specify how a dirty database should be updated to obtain a clean instance, especially when several *interacting updates* are required in order to enforce all the MDs. Thus, it does not give a complete prescription for the purpose of cleaning dirty instances. Moreover, from a different perspective, the requirements in the definition may be too strong, as the following example shows.

Example 2. Consider the set of MDs Σ consisting of φ_1: $R[A] \approx R[A] \rightarrow R[B] \rightleftharpoons R[B]$ and φ_2: $R[B, C] \approx R[B, C] \rightarrow R[D] \rightleftharpoons R[D]$. The similarities are: $a_1 \approx a_2$, $b_2 \approx b_3$, $c_2 \approx c_3$. Instance D_0 below is not a stable instance, i.e., it does not satisfy φ_1, φ_2. We start by enforcing φ_1 on D_0.

D_0	A	B	C	D
	a_1	b_1	c_1	d_1
	a_2	b_2	c_2	d_2
	a_3	b_3	c_3	d_3

D_1	A	B	C	D
	a_1	$\langle b_1, b_2\rangle$	c_1	d_1
	a_2	$\langle b_1, b_2\rangle$	c_2	d_2
	a_3	b_3	c_3	d_3

Let $\langle b_1, b_2\rangle$ in instance D_1 denote the value that replaces b_1 and b_2 to enforce φ_1 on instance D_0, and assume that $\langle b_1, b_2\rangle \not\approx b_3$. Now, $(D_0, D_1) \models \varphi_1$. However, $(D_0, D_1) \not\models \varphi_2$.

If we identify d_2, d_3 via $\langle d_2, d_3\rangle$ producing instance D_2, the pair (D_0, D_2) satisfies condition (a) in Definition 1 for φ_2, but not condition (b). Notice that making more updates on D_1 (or D_2) to obtain an instance D', such that $(D_0, D') \models \Sigma$, seems hopeless as φ_2 will not be satisfied because of the broken similarity that existed between b_2 and b_3. ■

Definition 1 seems to capture well the one-step enforcement of a single MD. However, as shown by the above example, the definition has to be refined in order to deal with sets of interacting MDs and to capture an iterative process of MD enforcement. We address this problem in Section 4.

Another issue worth mentioning is that stable instances D' for D and Σ are not subject to any form of minimality criterion on D' in relation with D. We would expect such an instance to be obtained via the enforcement of the MDs, without unnecessary changes. Unfortunately, this is not the case here: If in Example 2 we keep only φ_1, and in instance D_1 we change a_3 by an arbitrary value a_4 that is not similar to either a_1 or a_2, we obtain a stable instance with origin in D_0, but the change of a_3 is unjustified and unnecessary. We will also address this issue.

Following [18], we assume in the rest of this paper that each MD is of the form $R_1[X_1] \approx R_2[X_2] \rightarrow R_1[A_1] \rightleftharpoons R_2[A_2]$. That is, the right-hand side of each MD contains a pair of single attributes.

3. MATCHING FUNCTIONS AND SEMANTIC DOMINATION

In order to enforce a set of MDs (cf. Section 4) we need an operation that identifies two values whenever necessary. With this purpose in mind, we will assume that for each comparable pair of attributes A_1, A_2 with domain Dom_A, there is a binary *matching function* $m_A : Dom_A \times Dom_A \rightarrow Dom_A$, such that the value $m_A(a, a')$ is used to replace two values $a, a' \in Dom_A$ whenever the two values need to be made equal. Here are a few natural properties to expect of the matching function m_A (similar properties were considered in [8], cf. Section 7): For $a, a', a'' \in Dom_A$,

I (Idempotency): $m_A(a, a) = a$,
C (Commutativity): $m_A(a, a') = m_A(a', a)$,
A (Associativity): $m_A(a, m_A(a', a'')) = m_A(m_A(a, a'), a'')$.

It is reasonable to assume that any matching function satisfies at least these three axioms. Idempotency is a natural assumption as it is never desirable to replace two values that are already identical with a new value. Commutativity and associativity are also expected, intuitively because applying a matching function to make two or more values identical should not be sensitive to the order in which those values are visited. Under these assumptions, the structure (Dom_A, m_A) forms a *join semilattice*, L_A, that is, a partial order with a least upper bound (*lub*) for every pair of elements. The induced partial order $\preceq_A$ on the elements of Dom_A is defined as follows: For every $a, a' \in Dom_A$, $a \preceq_A a'$ whenever $m_A(a, a') = a'$. The *lub* operator with respect to this partial order coincides with m_A: $lub_{\preceq_A}\{a, a'\} = m_A(a, a')$.

A natural interpretation for the partial order $\preceq_A$ in the context of data cleaning would be the notion of *semantic domination*. Basically, for two elements $a, a' \in Dom_A$, we say that a' *semantically dominates* a if we have $a \preceq_A a'$.

[1] All the variables in X_i, Y_j are implicitly universally quantified in front of the formula.

In the process of cleaning the data by enforcing matching dependencies, we always replace two values a, a', whenever certain similarities hold, by the value $m_A(a, a')$ that semantically dominates both a and a'. This notion of domination is also related to relative information contents [12, 22, 23].

To define the semantics of query answering on instances that have been cleaned with matching dependencies, we might, in addition, need the existence of the greatest lower bound (*glb*) for any two elements in the domain of an attribute. We therefore assume that (Dom_A, m_A) is a lattice (i.e., both *lub* and *glb* exist for every pair of elements in Dom_A w.r.t. $\preceq_A$). Moreover, there is a special element $\perp \in Dom_A$ such that $m_A(a, \perp) = a$, for every $a \in Dom_A$. Notice that if we add an additional assumption to the semilattice, which requires that for every element $a \in Dom_A$, the set $\{c \in Dom_A \mid c \preceq_A a\}$ (the set of elements c with $m_A(a, c) = a$), is finite, then $glb_{\preceq_A}\{a, a'\}$ does exist for every two elements $a, a' \in Dom_A$ and is equal to $lub_{\preceq_A}\{c \in Dom_A \mid c \preceq_A a \text{ and } c \preceq_A a'\}$. We could also assume the existence of another special element $\top \in Dom_A$ such that $m_A(a, \top) = \top$, for every $a \in Dom_A$. This element could represent the existence of inconsistency in data whenever matching dependencies force to match two completely unrelated elements a, a' from the domain, in which case $m_A(a, a') = \top$. However, *the existence of $\top$ is not essential in our framework.*

Example 3. We give a few concrete examples of matching functions for different attribute domains. Our example functions have all the properties **I**, **C**, and **A**.

Name, Address, Phone Each atomic value s of these string domains could be treated as a singleton set $\{s\}$. Then a matching function $m(S_1, S_2)$ for sets of strings S_i could return $S_1 \cup S_2$. E.g., when matching addresses, $m(\{\text{'2366 Main Mall'}\}, \{\text{'Main Mall, Vancouver'}\})$ could return {'2366 Main Mall', 'Main Mall, Vancouver'}. (This union matching function is further investigated in Section 7.) Alternatively, a more sophisticated matching function could merge two input strings into a third string that contains both of the inputs. E.g., the match of the two input strings above could instead be '2366 Main Mall, Vancouver'.

Age, Salary, Price Each atomic value a in these numerical domains could be treated as an interval $[a, a]$. Then the matching function $m([a_1, b_1], [a_2, b_2])$ would return the smallest interval containing both $[a_1, b_1]$ and $[a_2, b_2]$, i.e., $m([a_1, b_1], [a_2, b_2]) = [min\{a_1, a_2\}, max\{b_1, b_2\}]$.

Boolean Attributes For attributes which take either a 0 or 1 value, the matching function would return $m(0, 1) = \top$, where $\top$ shows inconsistency in the data, and furthermore $m(0, \top) = \top$ and $m(1, \top) = \top$. In this case, the purpose of applying the matching function is to *record* the inconsistency in the data and still conduct sound reasoning in presence of inconsistency.[2] ■

An additional property of matching functions worthy of consideration is *similarity preservation*, that is, the result of applying a matching function preserves the similarity that existed between the old value being replaced and other values in the domain. More formally:

S (Similarity Preservation): If $a \approx a'$, then $a \approx m_A(a', a'')$, for every $a, a', a'' \in Dom_A$.

Unlike the previous properties (**I**, **C**, **A**), property **S** turns out to be a strong assumption, and we must consider both matching functions with **S** and without it. Indeed, notice that since $\approx$ subsumes equality, and m_A is commutative, assumption **S** implies $a \approx m_A(a, a')$ and $a' \approx m_A(a, a')$, i.e., similarity preserving matching always results in a value similar to the value being replaced. In the rest of the paper, we assume that for every comparable pair of attributes A_1, A_2, there is an idempotent, commutative, and associative binary matching function m_A. Unless otherwise specified, we do not assume that these functions are similarity preserving.

Definition 2. Let D_1, D_2 be instances of schema $\mathcal{R}$, and t_1, t_2 be two R-tuples in D_1, D_2, respectively, with $R \in \mathcal{R}$. We write $t_1^{D_1} \preceq t_2^{D_2}$ if $t_1^{D_1}[A] \preceq_A t_2^{D_2}[A]$ for every attribute A in R. We write $D_1 \sqsubseteq D_2$ if for every tuple t_1 in D_1, there is a tuple t_2 in D_2, such that $t_1^{D_1} \preceq t_2^{D_2}$. ■

Clearly, the relation $\preceq$ on tuples can be applied to tuples in the same instance. The ordering $\sqsubseteq$ on sets has been used in the context of complex objects [6, 25] and also powerdomains, where it is called *Hoare ordering* [12]. It is also used in [8] for entity resolution (cf. Section 7). It is known that for $\sqsubseteq$ to be a partial order, specifically to be antisymmetric, we need to deal with *reduced* instances [6], i.e., instances D in which there are no two different tuples t_1, t_2, such that $t_1^D \preceq t_2^D$. We can obtain the *reduced version* of every instance D by removing every tuple t_1, such that $t_1^D \preceq t_2^D$ for another tuple t_2 in D.

Next we will show that the set of reduced instances with the partial order $\sqsubseteq$ forms a lattice: the least upper bound and the greatest lower bound for every finite set of reduced instances exist. This result will be used later for query answering. We adapt some of the results from [6], where they prove a similar result for a lattice formed by the set of complex objects and the sub-object partial order.

Definition 3. Let D_1, D_2 be instances of schema $\mathcal{R}$, and t_1, t_2 be two R-tuples in D_1, D_2, respectively, for $R \in \mathcal{R}$.

(a) We define $D_1 \curlyvee D_2$ to be the reduced version of $D_1 \cup D_2$, where $D_1 \cup D_2$ refers to the set-theoretic union of D_1 and D_2.

(b) We define $t_1 \curlywedge t_2$ to be tuple t, such that $t[A] = glb_{\preceq_A}\{t_1^{D_1}[A], t_2^{D_2}[A]\}$ for every attribute A in R.

(c) We define $D_1 \curlywedge D_2$ to be the reduced version of the instance that assigns the set of tuples $\{t_1 \curlywedge t_2 \mid t_1 \in D_1,\ t_2 \in D_2, t_1, t_2 \text{ } R\text{-tuples}\}$ to every $R \in \mathcal{R}$. ■

Next we show that the operations defined in Definition 3 are equivalent to the greatest lower bound and least upper bound of instances w.r.t. the partial order $\sqsubseteq$.

Lemma 1. For every two instances D_1, D_2 and R-tuples t_1, t_2 in D_1, D_2, the following holds:

1. $D_1 \curlyvee D_2$ is the least upper bound of D_1, D_2 w.r.t. $\sqsubseteq$.
2. $t_1 \curlywedge t_2$ is the greatest lower bound of t_1, t_2 w.r.t. $\preceq$.

[2] Matching of boolean attributes requires the existence of the top element $\top$.

3. $D_1 \curlywedge D_2$ is the greatest lower bound of D_1, D_2 w.r.t. $\sqsubseteq$. ■

In particular, we can see that $\preceq$ imposes a lattice structure on R-tuples. Using Lemma 1, we immediately obtain the following result.

THEOREM 1. The set of reduced instances for a given schema with the $\sqsubseteq$ ordering forms a lattice. ■

4. ENFORCEMENT OF MDS AND CLEAN INSTANCES

In this section, we define *clean* instances that can be obtained from a dirty instance by iteratively enforcing a set of MDs in a chase-like procedure. Let D, D' be two database instances with the same set of tuple identifiers, and t_1, t_2 be an R_1-tuple and an R_2-tuple, respectively, in both D and D'. Let Σ be a set of MDs, and $\varphi : R_1[X_1] \approx R_2[X_2] \rightarrow R_1[A_1] \rightleftharpoons R_2[A_2]$ be an MD in Σ.

DEFINITION 4. Instance D' is the immediate result of enforcing φ on t_1, t_2 in instance D, denoted by $(D, D')_{[t_1,t_2]} \models \varphi$, if

1. $t_1^D[X_1] \approx t_2^D[X_2]$, but $t_1^D[A_1] \neq t_2^D[A_2]$;
2. $t_1^{D'}[A_1] = t_2^{D'}[A_2] = m_A(t_1^D[A_1], t_2^D[A_2])$; and
3. D, D' agree on every other tuple and attribute value.

■

Definition 4 captures a single step in a chase-like procedure that starts from a dirty instance D_0 and enforces MDs step by step, by applying matching functions, until the instance becomes stable. We propose that the output of this chase should be treated as a clean version of the original instance, given a set of MDs, formally defined as follows.

DEFINITION 5. For an instance D_0 and a set of MDs Σ, an instance D_k is (D_0, Σ)-*clean* if D_k is stable, and there exists a finite sequence of instances $D_1, \ldots, D_{k-1}$ such that, for every $i \in [1, k]$, $(D_{i-1}, D_i)_{[t_1^i, t_2^i]} \models \varphi^i$, for some $\varphi^i \in \Sigma$ and tuple identifiers t_1^i, t_2^i. ■

Notice that if $(D_0, D_0) \models \Sigma$, i.e., it is already stable, then D_0 is its only (D_0, Σ)-clean instance. Moreover, we have $D_{i-1} \sqsubseteq D_i$, for every $i \in [1, k]$, since we are using matching functions to identify values. In particular, we have $D_0 \sqsubseteq D_k$. In other words, clean instance D_k semantically dominates dirty instance D_0, and we might say D_k it is *more informative* than D_0.

THEOREM 2. Let Σ be a set of matching dependencies and D_0 be an instance. Then every sequence $D_1, D_2, \ldots$ such that $(D_{i-1}, D_i)_{[t_1^i, t_2^i]} \models \varphi^i$, for some $\varphi^i \in \Sigma$ and tuple identifiers t_1^i, t_2^i in D_{i-1}, is finite and computes a (D_0, Σ)-clean instance D_k in polynomial number of steps in the size of D_0.

Proof sketch. It is easy to see that for every $i \in [1, k]$, we have $D_{i-1} \sqsubseteq D_i$ and $D_i \not\sqsubseteq D_{i-1}$. That is, D_i strictly dominates D_{i-1} (for simplicity, assume that the new generated tuples are not completely dominated by other tuples). Consider a database instance D_{max} that has a single tuple in every relation, for which the value of every attribute A is the result of matching all values of A (and other attributes comparable with A) in the active domain of D_0. Clearly, D_{max} provides an upper bound for the instances in the sequence, and thus the sequence stops after finite number of steps. Furthermore, the number of matching applications needed to reach D_{max} is polynomial in the size of D_0. ■

In other words, the sequence of instances obtained by chasing MDs reaches a fixpoint after polynomial number of steps. That is, it is not possible to generate a new instance, because condition 1 in Definition 4 is not satisfied by the last generated instance, i.e., the last instance is stable w.r.t. all MDs. This is the consequence of assuming that matching functions are idempotent, commutative, and associative.

Observe that, for a given instance D_0 and set of MDs Σ, (finitely) many clean instances may exist, each resulting from a different order of application of MDs on D_0 and from different selections of violating tuples.

Notice also that for a (D_0, Σ)-clean instance D_k, we may have $(D_0, D_k) \not\models \Sigma$ (cf. Definition 1). Intuitively, the reason is that some of the similarities that existed in D_0 could have been broken by iteratively enforcing the MDs to produce D_k. We argue that this is a price we may have to pay if we want to enforce a set of interacting MDs. However, each (D_0, Σ)-clean instance is stable and captures the persistence of attribute values that are not affected by MDs. The following example illustrates these points. For simplicity, we write $\langle a_1, \ldots, a_l \rangle$ to represent $m_A(a_1, m_A(a_2, m_A(\ldots, a_l)))$, which is allowed by the associativity assumption.

EXAMPLE 4. Consider the set of MDs Σ consisting of φ_1: $R[A] \approx R[A] \rightarrow R[B] \rightleftharpoons R[B]$ and φ_2: $R[B] \approx R[B] \rightarrow R[C] \rightleftharpoons R[C]$. We have the similarities: $a_1 \approx a_2$, $b_2 \approx b_3$. The following sequence of instances leads to a (D_0, Σ)-clean instance D_2.

D_0	A	B	C
	a_1	b_1	c_1
	a_2	b_2	c_2
	a_3	b_3	c_3

D_1	A	B	C
	a_1	$\langle b_1, b_2 \rangle$	c_1
	a_2	$\langle b_1, b_2 \rangle$	c_2
	a_3	b_3	c_3

D_2	A	B	C
	a_1	$\langle b_1, b_2 \rangle$	$\langle c_1, c_2 \rangle$
	a_2	$\langle b_1, b_2 \rangle$	$\langle c_1, c_2 \rangle$
	a_3	b_3	c_3

However, $(D_0, D_2) \not\models \Sigma$, and the reason is that $\langle b_1, b_2 \rangle \approx b_3$ does not necessarily hold. We can enforce the MDs in another order and obtain a different (D_0, Σ)-clean instance:

D_0	A	B	C
	a_1	b_1	c_1
	a_2	b_2	c_2
	a_3	b_3	c_3

D_1'	A	B	C
	a_1	b_1	c_1
	a_2	b_2	$\langle c_2, c_3 \rangle$
	a_3	b_3	$\langle c_2, c_3 \rangle$

D_2'	A	B	C
	a_1	$\langle b_1, b_2 \rangle$	c_1
	a_2	$\langle b_1, b_2 \rangle$	$\langle c_2, c_3 \rangle$
	a_3	b_3	$\langle c_2, c_3 \rangle$

D_3'	A	B	C
	a_1	$\langle b_1, b_2 \rangle$	$\langle c_1, c_2, c_3 \rangle$
	a_2	$\langle b_1, b_2 \rangle$	$\langle c_1, c_2, c_3 \rangle$
	a_3	b_3	$\langle c_2, c_3 \rangle$

Again, D_3' is a (D_0, Σ)-clean instance, but $(D_0, D_3') \not\models \Sigma$. ■

It would be interesting to know when there is only one (D_0, Σ)-clean instance D_k, and also when, for a clean instance D_k, $(D_0, D_k) \models \Sigma$ holds. The following two propo-

sitions establish natural sufficient conditions for these properties to hold.

PROPOSITION 1. Suppose that for every pair of comparable attributes A_1, A_2, the matching function m_A is similarity preserving. Then, for every set of MDs Σ and every instance D_0, there is a unique (D_0, Σ)-clean instance D_k. Furthermore, $(D_0, D_k) \models \Sigma$.

Proof sketch. Let D, D' be two (D_0, Σ)-clean instances, and let $D_1, \ldots, D_k$ be a sequence of instances such that $D = D_k$, and for every $i \in [1, k]$, $(D_{i-1}, D_i)_{[t_1^i, t_2^i]} \models \varphi^i$, for some $\varphi^i \in \Sigma$ and tuple identifiers t_1^i, t_2^i. We can show that D, D' are equivalent by showing that for every $i \in [0, k]$, the following holds (using an induction on i):

1. $t_1^{D_i}[A] \preceq_A t_1^{D'}[A]$, for every tuple identifier t_1 and every attribute A.

2. If $t_1^{D_i}[A_1] \approx t_2^{D_i}[A_2]$, then $t_1^{D'}[A_1] \approx t_2^{D'}[A_2]$, for every two tuple identifiers t_1, t_2 and two comparable attributes A_1, A_2. ■

We say that a set of matching dependencies Σ is *interaction-free* if for every two MDs $\varphi_1, \varphi_2 \in \Sigma$, the two sets of attributes on the right-hand side of φ_1 and left-hand side of φ_2 are disjoint. The two sets of MDs in Examples 2 and 4 are not interaction-free.

PROPOSITION 2. Let Σ be an interaction-free set of MDs. Then for every instance D_0, there is a unique (D_0, Σ)-clean instance D_k. Furthermore, $(D_0, D_k) \models \Sigma$. ■

The chase-like procedure that produces a (D_0, Σ)-clean instance makes only those changes to instance D_0 that are necessary, and are imposed by the dynamic semantics of MDs. In this sense, we can say that the chase implements minimal changes necessary to obtain a clean instance.

Another interesting question is whether (D_0, Σ)-clean instances are at a minimal distance to D_0 w.r.t. the partial order $\sqsubseteq$. This is not true in general. For instance in Example 4, observe that for the two (D_0, Σ)-clean instances D_2 and D_3', $D_2 \sqsubseteq D_3'$, but $D_3' \not\sqsubseteq D_2$, which means D_3' is not at a minimal distance to D_0 w.r.t. $\sqsubseteq$. We have actually no reason to expect the clean instances to be minimal in this sense since they are obtained as fixpoints of two different chase paths. However, both of these clean instances may be useful in query answering, because, informally speaking, they can provide a lower bound and an upper bound for the possible clean interpretations of the original dirty instance w.r.t. the semantic domination. This issue is discussed in the next section.

5. CLEAN QUERY ANSWERING

Most of the literature on data cleaning has concentrated on producing a clean instance starting from a dirty one. However, the problem of characterizing and retrieving the data in the original instance that can be considered to be clean has been neglected. In this section we study this problem, focusing on query answering. More precisely, given an instance D, a set Σ of MDs, and a query $\mathcal{Q}$ posed to D, we want to characterize the answers that are consistent with Σ, i.e., that would be returned by an instance where all the MDs have been enforced. Of course, we have to take into account that there may be several such instances.

This situation is similar to the one encountered in *consistent query answering* (CQA) [3, 9, 14], where query answering is characterized and performed on database instances that may fail to satisfy certain classic integrity constraints (ICs). For such a database instance, a *repair* is an instance that satisfies the integrity constraints and minimally differs from the original instance. For a given query, a *consistent answer* (a form of certain answer) is defined as the set of tuples that are present in the intersection of answers to the query when posed to every repair. A less popular alternative is the notion of *possible answer*, that is defined as the union of all tuples that are present in the answer to the query when posed to every repair.

A similar semantics for clean query answering under matching dependencies can be defined. However, the partial order relationship $\sqsubseteq$ between a dirty instance and its clean instances establishes an important difference between clean instances w.r.t. matching dependencies and repairs w.r.t. traditional ICs.

Intuitively, a clean instance has improved the information that already existed in the dirty instance and made it more informative and consistent. We would like to carefully take advantage of this partial order relationship and use it in the definition of certain and possible answers. We do this by taking the greatest lower bound (glb) and least upper bound (lub) of answers of the query over multiple clean instances, instead of taking the set-theoretic intersection and union.

Let Σ be a set of MDs, D_0 be a database instance, and $\mathcal{Q}$ be a query posed to instance D_0. We define *certain* and *possible answers* as follows.

$$Cert_{\mathcal{Q}}(D_0) = glb_{\sqsubseteq}\{\mathcal{Q}(D) \mid D \text{ is a } (D_0, \Sigma)\text{-clean inst.}\}. \quad (2)$$

$$Poss_{\mathcal{Q}}(D_0) = lub_{\sqsubseteq}\{\mathcal{Q}(D) \mid D \text{ is a } (D_0, \Sigma)\text{-clean inst.}\}. \quad (3)$$

The glb and lub above are defined on the basis of the partial order $\sqsubseteq$ on sets of tuples. Since there is a finite number of clean instances for D_0, these glb and lub exist (cf. Theorem 1). In Eq. (2) and (3) we are assuming that each $\mathcal{Q}(D)$ is reduced (cf. Section 3). By Definition 3, $Cert_{\mathcal{Q}}(D_0)$ and $Poss_{\mathcal{Q}}(D_0)$ are also reduced. Moreover, we clearly have $Cert_{\mathcal{Q}}(D_0) \sqsubseteq Poss_{\mathcal{Q}}(D_0)$.

The following example motivates these choices. It also shows that, unlike some cases of inconsistent databases and consistent query answering, certain answers could be quite informative and meaningful for databases with matching dependencies.

EXAMPLE 5. Consider relation $R(name, phone, address)$, and set Σ consisting of the following MDs:

φ_1 : $R[name, phone, address] \approx R[name, phone, address] \rightarrow R[address] \rightleftharpoons R[address]$,
φ_2 : $R[phone, address] \approx R[phone, address] \rightarrow R[phone] \rightleftharpoons R[phone]$.

Suppose that in the dirty instance D_0, shown below, the following similarities hold:

"John Doe" ≈ "J. Doe",
"Jane Doe" ≈ "J. Doe",
"(613)123 4567" ≈ "123 4567",
"(604)123 4567" ≈ "123 4567",
"25 Main St." ≈ "Main St., Ottawa",
"25 Main St." ≈ "25 Main St., Vancouver".

Other non-trivial similarities that are not mentioned do not hold. Moreover, the matching functions act as follows:

m_{phone}("(613)123 4567", "123 4567") = "(613)123 4567",
m_{phone}("123 4567", "(604)123 4567") = "(604)123 4567",
$m_{address}$("Main St., Ottawa", "25 Main St.") =
"25 Main St., Ottawa",
$m_{address}$("25 Main St.", "25 Main St., Vancouver") =
"25 Main St., Vancouver".

D_0	*name*	*phone*	*address*
	John Doe	(613)123 4567	Main St., Ottawa
	J. Doe	123 4567	25 Main St.
	Jane Doe	(604)123 4567	25 Main St., Vancouver

Observe that from D_0 we can obtain two different (D_0, Σ)-clean instances D, D', depending on the order of enforcing MDs on different pairs of tuples.

D	*name*	*phone*	*address*
	John Doe	(613)123 4567	25 Main St., Ottawa
	J. Doe	(613)123 4567	25 Main St., Ottawa
	Jane Doe	(604)123 4567	25 Main St., Vancouver

D'	*name*	*phone*	*address*
	John Doe	(613)123 4567	Main St., Ottawa
	J. Doe	(604)123 4567	25 Main St., Vancouver
	Jane Doe	(604)123 4567	25 Main St., Vancouver

Now consider the query $\mathcal{Q} : \pi_{address}(\sigma_{name=\text{"J. Doe"}}R)$, asking for the residential address of J. Doe. We are interested in a certain answer. It can be obtained by taking the greatest lower bound of the two answer sets:

$$\mathcal{Q}(D) = \{(\text{"25 Main St., Ottawa"})\},$$
$$\mathcal{Q}(D') = \{(\text{"25 Main St., Vancouver"})\}.$$

In this case, and according to [6], and using Lemma 1,

$glb_{\sqsubseteq}\{\mathcal{Q}(D), \mathcal{Q}(D')\} = \{a \curlywedge a' \mid a \in \mathcal{Q}(D),\ a' \in \mathcal{Q}(D')\}$
$= \{(\text{"25 Main St., Ottawa"}) \curlywedge (\text{"25 Main St., Vancouver"})\}$
$= \{(glb_{\preceq_{address}}\{\text{"25 Main St., Ottawa"}, \text{"25 Main St., Vancouver"}\})\}$
$= \{(\text{"25 Main St."})\}.$

We can see that, no matter how we clean D_0, we can say for sure that J. Doe is at 25 Main St. Notice that the set-theoretic intersection of the two answer sets is empty. If we were interested in all possible answers, we could take the least upper bound of two answer sets, which would be the union of the two in this case. ■

We define *clean answer* to be an upper and lower bound of query answers over all possible clean interpretations of a dirty database instance. This definition is inspired by the same kind of approximations used in the contexts of partial and incomplete information [30, 1], inconsistent databases [3, 9, 14], and data exchange [29]. These upper and lower bounds could provide useful information about the value of aggregate functions, such as sum and count [5, 19, 2].

Definition 6. For a query $\mathcal{Q}$ posed to a database instance D_0 and a set of MDs Σ, a *clean answer* is specified by two bounds as

$$Clean_{\mathcal{Q}}(D_0) = (Cert_{\mathcal{Q}}(D_0), Poss_{\mathcal{Q}}(D_0)). \quad ■$$

Notice that in the case of having similarity-preserving matching functions or non-interacting matching dependencies, from the results in Section 4, these bounds would collapse into a single set, which is obtained by running the query on the unique clean instance.

5.1 Complexity of Computing Clean Answers

Here we study the complexity of computing clean answers over database instances in presence of MDs. As with incomplete and inconsistent databases, this problem easily becomes intractable for simple MDs and queries, which motivates the need for developing approximate solutions to these problems. We explore approximate solutions for queries that behave monotonically w.r.t. the partial order $\sqsubseteq$ in Section 6.2.

Theorem 3. There are a schema with two interacting MDs and a relational algebra query, for which deciding whether a tuple belongs to the certain answer set for an instance D_0 is coNP-complete (in the size of D_0).

Proof sketch. Consider relation schema $R(C, V, L)$, query $\mathcal{Q} : \pi_L(R)$, and set Σ consisting of two MDs $\varphi_1 : R[C] \approx R[C] \rightarrow R[C] \rightleftharpoons R[C]$ and $\varphi_2 : R[CV] \approx R[CV] \rightarrow R[L] \rightleftharpoons R[L]$. A simple way to establish membership in coNP would be to slightly modify the definition of the chase procedure and clean instance by associating with each tuple an annotation that keeps track of its evolution, in terms of the MDs applied and the other tuples participating in the matching. Having explicit chase history as part of the annotation would allow us to check in polynomial time whether a given witness instance D is indeed a (D_0, Σ)-clean instance, and also to check whether a given tuple is not in $\mathcal{Q}(D)$, and thus is not in the certain answer. To prove hardness, we translate an instance $\mathbf{C}$ of the 3SAT problem into an instance D_0 of $R(C, V, L)$, and show that formula $\mathbf{C}$ is satisfiable if and only if $\top \notin Cert_{\mathcal{Q}}(D_0)$. ■

6. MONOTONE QUERIES

So far we have seen that clean instances are a more informative view of a dirty instance obtained by enforcing matching dependencies. That is, $D_0 \sqsubseteq D$, for every (D_0, Σ)-clean instance D. From this perspective, it would be natural to expect that for a positive query, we would obtain a more informative answer if we pose it to a clean instance instead of to the dirty one. We can translate this requirement into a monotonicity property for queries w.r.t. the partial order $\sqsubseteq$.

Definition 7. A query $\mathcal{Q}$ is $\sqsubseteq$-*monotone* if, for every pair of instances D, D', such that $D \sqsubseteq D'$, we have $\mathcal{Q}(D) \sqsubseteq \mathcal{Q}(D')$. ■

Monotone queries have an interesting behavior when computing clean answers. For these queries, we can under-approximate (over-approximate) certain answers (possible answers) by taking the greatest lower bound (least upper bound) of all clean instances and then running the query on the result. Notice that we are not claiming that these are polynomial-time approximations.

Proposition 3. If $\mathcal{D}$ is a finite set of database instances and $\mathcal{Q}$ is a $\sqsubseteq$-monotone query, the following holds:

$$\mathcal{Q}(glb_{\sqsubseteq}\{D \mid D \in \mathcal{D}\}) \sqsubseteq glb_{\sqsubseteq}\{\mathcal{Q}(D) \mid D \in \mathcal{D}\}, \quad (4)$$

$$lub_{\sqsubseteq}\{\mathcal{Q}(D) \mid D \in \mathcal{D}\} \sqsubseteq \mathcal{Q}(lub_{\sqsubseteq}\{D \mid D \in \mathcal{D}\}). \quad (5)$$

■

As is well known, positive relational algebra queries composed of selection, projection, Cartesian product, and union, are monotone w.r.t. $\subseteq$. However, the following example shows that monotonicity w.r.t. $\sqsubseteq$ does not hold even for very simple positive queries involving selections.

EXAMPLE 6. Consider instance D_0 in Example 5 and the two (D_0, Σ)-clean instances D and D'. Let $\mathcal{Q}$ be a query asking for names of people residing at "25 Main St.", expressed in relational algebra as $\pi_{name}(\sigma_{address=\text{"25 Main St."}}(R))$. Observe that $\mathcal{Q}(D_0) = \{(\text{"J. Doe"})\}$, and $\mathcal{Q}(D) = \mathcal{Q}(D') = \emptyset$. Query $\mathcal{Q}$ is therefore not $\sqsubseteq$-monotone, because we have $D_0 \sqsubseteq D$, $D_0 \sqsubseteq D'$, but $\mathcal{Q}(D_0) \not\sqsubseteq \mathcal{Q}(D)$, $\mathcal{Q}(D_0) \not\sqsubseteq \mathcal{Q}(D')$. ■

It is not surprising that $\sqsubseteq$-monotonicity is not satisfied by usual relational queries, in particular, by queries that *are* monotone w.r.t. set inclusion. After all, the queries we have considered so far do not even mention the $\preceq$ predicate that is at the basis of the $\sqsubseteq$ order. Next we will consider queries expressing conditions in terms of the semantic domination lattice to achieve $\sqsubseteq$-monotonicity (see Example 7).

6.1 Query relaxation

As shown in Example 6, we may not get the answer we expect by running a usual relational algebra query on an instance that has been cleaned using matching dependencies. We therefore propose to *relax* the queries, by taking advantage of the underlying $\preceq$-lattice structure obtained from matching functions, to make them $\sqsubseteq$-monotone. In this way, we achieve two goals: First, the resulting queries provide more informative answers; and second, we can take advantage of Proposition 3 to approximate clean answers from below and from above.

We introduce the (negation free) language *relaxed relational algebra*, $\mathcal{RA}_{\preceq}$, by providing two selection operators $\sigma_{a \preceq A}$ and $\sigma_{A_1 \bowtie_{\preceq} A_2}$ (for comparable attributes A_1, A_2), defined as follows.

DEFINITION 8. The language $\mathcal{RA}_{\preceq}$ is composed of relational operators $\pi, \times, \cup$ (with usual definitions), plus $\sigma_{a \preceq A}$, and $\sigma_{A_1 \bowtie_{\preceq} A_2}$, defined by:

$$\sigma_{a \preceq A}(D) = \{t^D \mid a \preceq_A t^D[A]\} \quad (\text{here } a \in Dom_A),$$
$$\sigma_{A_1 \bowtie_{\preceq} A_2}(D) = \{t^D \mid \exists a \in Dom_A \text{ s.t. } a \preceq_A t^D[A_1], \; a \preceq_A t^D[A_2], \; a \neq \bot\}.$$

■

For string attributes, for instance, the selection operator $\sigma_{a \preceq A}$ checks whether the value of attribute A dominates the substring a, and the join selection operator $\sigma_{A_1 \bowtie_{\preceq} A_2}$ checks whether the values of attributes A_1, A_2 dominate a common substring. Notice that queries in the language $\mathcal{RA}_{\preceq}$ are not domain independent: The result of posing a query to an instance depends not only on the values in the active domain of the instance but also on the domain lattices. In other words, query answering depends on how data cleaning is being implemented.

It can be easily observed that all operators in the language $\mathcal{RA}_{\preceq}$ are $\sqsubseteq$-monotone, and therefore every query expression in $\mathcal{RA}_{\preceq}$ that is obtained by composing these operators is also $\sqsubseteq$-monotone.

PROPOSITION 4. Let $\mathcal{Q}$ be a query in $\mathcal{RA}_{\preceq}$. For every two instances D, D' such that $D \sqsubseteq D'$, we have $\mathcal{Q}(D) \sqsubseteq \mathcal{Q}(D')$. ■

Now suppose that we have a query $\mathcal{Q}$, written in positive relational algebra, i.e., composed of $\pi, \times, \cup, \sigma_{A=a}, \sigma_{A_1=A_2}$, the last two being *hard* selection conditions, which is to be posed to an instance D_0. After cleaning D_0 by enforcing a set of MDs Σ to obtain a (D_0, Σ)-clean instance D, running query $\mathcal{Q}$ on D may no longer provide the expected answer, because some of the values have changed in D, i.e., they have semantically grown w.r.t. $\preceq$. In order to capture this semantic growth, our query relaxation framework proposes the following *query rewriting* methodology: Given a query $\mathcal{Q}$ in positive RA, transform it into a query $\mathcal{Q}_{\preceq}$ in $\mathcal{RA}_{\preceq}$ by simply replacing the selection operators $\sigma_{A=a}$ and $\sigma_{A_1=A_2}$ by $\sigma_{a \preceq A}$ and $\sigma_{A_1 \bowtie_{\preceq} A_2}$, respectively.

EXAMPLE 7. Consider again instance D_0 in Example 5 and (D_0, Σ)-clean instances D and D', and query $\mathcal{Q}$ asking for names of people residing at "25 Main St.", expressed as $\pi_{name}(\sigma_{address=\text{"25 Main St."}}(R))$. We obtain the empty answer from each of D, D'. So, in this case the certain and the possible answers are empty, a not very informative outcome.

However, after the relaxation rewriting of $\mathcal{Q}$, we obtain the monotone query $\mathcal{Q}_{\preceq}$: $\pi_{name}(\sigma_{\text{"25 Main St."} \preceq address}(R))$. If we pose $\mathcal{Q}_{\preceq}$ to the clean instances, we obtain

$$\mathcal{Q}_{\preceq}(D) = \{(\text{"John Doe"}), (\text{"J. Doe"}), (\text{"Jane Doe"})\},$$
$$\mathcal{Q}_{\preceq}(D') = \{(\text{"J. Doe"}), (\text{"Jane Doe"})\},$$

and thus $Cert_{\mathcal{Q}_{\preceq}}(D_0) = \{(\text{"J. Doe"}), (\text{"Jane Doe"})\}$. This outcome is much more informative; and, above all, is sensitive to the underlying information lattice. ■

PROPOSITION 5. For every positive relational algebra query $\mathcal{Q}$ and every instance D, we have $\mathcal{Q}(D) \sqsubseteq \mathcal{Q}_{\preceq}(D)$, where $\mathcal{Q}_{\preceq}$ is the relaxed rewriting of $\mathcal{Q}$. ■

6.2 Approximating Clean Answers

Given the high computational cost of clean query answering when there are multiple clean instances, it would be desirable to provide an approximation to clean answers that is computable in polynomial time. In this section, we are interested in approximating clean answers by producing an under-approximation of certain answers and an over-approximation of possible answers for a given monotone query $\mathcal{Q}$. That is, we would like to obtain $(\mathcal{Q}_{\downarrow}(D_0), \mathcal{Q}_{\uparrow}(D_0))$, such that $\mathcal{Q}_{\downarrow}(D_0) \sqsubseteq Cert_{\mathcal{Q}}(D_0)$ and $Poss_{\mathcal{Q}}(D_0) \sqsubseteq \mathcal{Q}_{\uparrow}(D_0)$.

Since $\mathcal{Q}$ is a monotone query, by Proposition 3, we have $\mathcal{Q}(glb_{\sqsubseteq}\{D \mid D \text{ is } (D_0, \Sigma)\text{-clean}\}) \sqsubseteq Cert_{\mathcal{Q}}(D_0)$, and moreover, $Poss_{\mathcal{Q}}(D_0) \sqsubseteq \mathcal{Q}(lub_{\sqsubseteq}\{D \mid D \text{ is } (D_0, \Sigma)\text{-clean}\})$. In consequence, it is good enough to find under- and over-approximations for the greatest lower bound and the least upper bound, resp., of the set of all (D_0, Σ)-clean instances, and then pose $\mathcal{Q}$ to these approximations to obtain $\mathcal{Q}_{\downarrow}(D_0)$ and $\mathcal{Q}_{\uparrow}(D_0)$.

The reason for having multiple clean instances is that matching dependencies are not necessarily interaction-free and the matching functions are not necessarily similarity preserving. Intuitively speaking, we can under-approximate the greatest lower bound of clean instances by not enforcing some of the interacting MDs. On the other side, we can over-approximate the least upper bound by assuming that the matching functions are similarity preserving. This would

lead us to keep applying MDs on the assumption that unresolved similarities still persist. We present two chase-like procedures to compute two instances $D_\downarrow$ and $D_\uparrow$ corresponding to these approximations.

Under-approximating the greatest lower bound.

To provide an under-approximation for the greatest lower bound of all clean instances, we provide a new chase-like procedure, which enforces only MDs that are enforced in every clean instance. These MDs are applicable to those initial similarities that exist in the original dirty instance, which are never broken by enforcing other MDs during any chase procedure of producing a clean instance.

Let Σ be a set of MDs, and $\varphi, \varphi' \in \Sigma$. We say that φ *precedes* φ' if the set of attributes on the left-hand side of φ' contains the attribute on the right-hand side of φ. We say that φ *interacts with* φ' if there are MDs $\varphi_1, \ldots, \varphi_k \in \Sigma$, such that φ precedes φ_1, φ_k precedes φ', and φ_i precedes φ_{i+1} for $i \in [1, k-1]$, i.e., the interaction relationship can be seen as the transitive closure of precedence relationship.

Let D_0 be a dirty database instance. Let $\varphi : R_1[X_1] \approx R_2[X_2] \rightarrow R_1[A_1] \rightleftharpoons R_2[A_2]$ be an MD in Σ. We say φ is *freshly applicable* on t_1, t_2 in D_0 if $t_1^{D_0}[X_1] \approx t_2^{D_0}[X_2]$, and $t_1^{D_0}[A_1] \neq t_2^{D_0}[A_2]$. We say φ is *safely applicable* on t_1, t_2 in D_0 if φ is freshly applicable on t_1, t_2 in D_0, and for every $\varphi' \in \Sigma$ that interacts with φ, φ' is not freshly applicable on t_1, t_3 or t_2, t_3 in D_0 for any tuple t_3 (see Example 8).

Definition 9. For an instance D_0 and a set of MDs Σ, an instance D_k is (D_0, Σ)-*under clean* if there exists a finite sequence of instances $D_1, \ldots, D_{k-1}$, such that

1. For every $i \in [1, k]$, $(D_{i-1}, D_i)_{[t_1^i, t_2^i]} \models \varphi^i$, for some $\varphi^i \in \Sigma$ and tuple identifiers t_1^i, t_2^i, such that φ^i is safely applicable on t_1^i, t_2^i in D_0.
2. For every MD $\varphi : R_1[X_1] \approx R_2[X_2] \rightarrow R_1[A_1] \rightleftharpoons R_2[A_2]$ in Σ and tuples t_1, t_2, such that φ is safely applicable on t_1, t_2 in D_0, we have $t_1^{D_k}[A_1] = t_2^{D_k}[A_2]$. ■

Definition 9 characterizes a chase-based procedure that keeps enforcing MDs that are safely applicable in the original dirty instance until all such MDs are enforced. Notice that an under clean instance may not be stable. Moreover, safely applicable MDs never interfere with each other, in the sense that enforcing one of them never breaks the initial similarities in the dirty instance that are needed for enforcing other safely applicable MDs.

Proposition 6. For every instance D_0 and every set of MDs Σ, there is a unique (D_0, Σ)-under clean instance $D_\downarrow$. ■

Clearly, an under clean instance $D_\downarrow$ can be computed in polynomial time in the size of the dirty instance D_0. To construct it, we first need to identify safely applicable MDs in D_0, and then enforce them in any arbitrary order until no such MDs can be enforced. Next we show that $D_\downarrow$ is an under-approximation to every (D_0, Σ)-clean instance. Intuitively, this is because $D_\downarrow$ is obtained by enforcing MDs that are enforced in every chase-based procedure of producing a clean instance.

Proposition 7. (*Soundness of under-approximation*) For every (D_0, Σ)-under clean instance $D_\downarrow$ and every (D_0, Σ)-clean instance D, we have $D_\downarrow \sqsubseteq D$. ■

Notice that an arbitrary (D_0, Σ)-clean instance D may not be a sound under-approximation for every other (D_0, Σ)-clean instances D', because $D \sqsubseteq D'$ may not hold.

Let $D_\downarrow$ be a (D_0, Σ)-under clean instance. Then from Propositions 7 and 3, we immediately obtain the following result.

Theorem 4. For every monotone query Q, we have $\mathcal{Q}(D_0) \sqsubseteq \mathcal{Q}(D_\downarrow) \sqsubseteq \mathit{Cert}_\mathcal{Q}(D_0)$. ■

Example 8. Consider the instance D_0 and set of MDs Σ in Example 4. Observe that MD φ_1 is safely applicable on the first and second tuples in D_0. Moreover, φ_2 is freshly applicable, but not safely applicable on the second and third tuples. Accordingly, we obtain (D_0, Σ)-under clean instance $D_\downarrow$, shown below, by enforcing φ_1 on the first two tuples.

$D_\downarrow$	A	B	C
	a_1	$\langle b_1, b_2 \rangle$	c_1
	a_2	$\langle b_1, b_2 \rangle$	c_2
	a_3	b_3	c_3

Notice that for the two (D_0, Σ)-clean instances D_2, D_3' in Example 4, we have $D_\downarrow \sqsubseteq D_2$ and $D_\downarrow \sqsubseteq D_3'$. Also notice that $D_\downarrow$ is not a stable instance. Now consider the query $\mathcal{Q} : \pi_C(\sigma_{A=a_2} R)$. This query behaves monotonically for our purpose, because the values of attribute A are not changing by enforcing MDs. If we pose $\mathcal{Q}$ to $D_\downarrow$, we obtain $\mathcal{Q}(D_\downarrow) = \{c_2\}$. Observe that $\mathit{Cert}_\mathcal{Q}(D_0) = \{\langle c_1, c_2 \rangle\}$, and thus $\mathcal{Q}(D_\downarrow)$ provides an under-approximation for $\mathit{Cert}_\mathcal{Q}(D_0)$. This example also shows that an arbitrary clean instance, D_3' here, may not provide a sound approximation to certain answer since $\mathcal{Q}(D_3') = \{\langle c_1, c_2, c_3 \rangle\} \not\sqsubseteq \mathit{Cert}_\mathcal{Q}(D_0)$. ■

Over-approximating the least upper bound.

To provide an over-approximation for the least upper bound of all clean instances, we *modify* every similarity relation so that the corresponding matching function becomes similarity preserving. For a similarity relation $\approx_A$ and the corresponding matching function m_A, we define $\approx_A^*$ as follows: For every $a, a' \in \mathit{Dom}_A$, $a \approx_A^* a'$ iff there is $a'' \in \mathit{Dom}_A$, such that $a \approx_A a''$ and $m_A(a', a'') = a'$. Given a set of MDs Σ, we obtain Σ^* by replacing every similarity relation $\approx_A$ in the MDs by $\approx_A^*$.

Definition 10. For an instance D_0 and a set of MDs Σ, an instance $D_\uparrow$ is (D_0, Σ)-*over clean* if $D_\uparrow$ is (D_0, Σ^*)-clean. ■

By Proposition 1, for every instance D_0 and set of MDs Σ^* involving only similarity preserving matching functions, there is a unique (D_0, Σ^*)-clean instance $D_\uparrow$, which is the (D_0, Σ)-over clean instance. Moreover, $D_\uparrow$ can be computed in polynomial time in the size of D_0. To construct $D_\uparrow$, we first need to obtain Σ^*, as described above, and enforce MDs in Σ^* in any arbitrary order until we get a stable instance w.r.t. Σ^*. Next we show that $D_\uparrow$ is an over-approximation for every (D_0, Σ)-clean instance. Intuitively, this is because $D_\uparrow$ is obtained by enforcing (at least) all MDs that are present in any chase-like procedure of producing a clean instance.

PROPOSITION 8. (*Completeness of over-approximation*) For every (D_0, Σ)-over clean instance $D_\uparrow$ and every (D_0, Σ)-clean instance D, we have $D \sqsubseteq D_\uparrow$. ■

Notice again that an arbitrary (D_0, Σ)-clean instance D may not be an over-approximation for every other (D_0, Σ)-clean instance D', because $D' \sqsubseteq D$ may not hold.

Let $D_\uparrow$ be a (D_0, Σ)-over clean instance. Then from Propositions 8 and 3, we immediately obtain the following result.

THEOREM 5. For every monotone query Q, we have $Poss_{\mathcal{Q}}(D_0) \sqsubseteq \mathcal{Q}(D_\uparrow)$. ■

EXAMPLE 9. (Example 8 continued.) By assuming that old similarities hold after applying matching functions (e.g., $\langle b_1, b_2\rangle \approx^* b_3$), we obtain the (D_0, Σ)-over clean instance $D_\uparrow$ shown below.

$D_\uparrow$	A	B	C
	a_1	$\langle b_1, b_2\rangle$	$\langle c_1, c_2, c_3\rangle$
	a_2	$\langle b_1, b_2\rangle$	$\langle c_1, c_2, c_3\rangle$
	a_3	b_3	$\langle c_1, c_2, c_3\rangle$

Notice that for the two (D_0, Σ)-clean instances D_2, D_3' in Example 4, we have $D_2 \sqsubseteq D_\uparrow$ and $D_3' \sqsubseteq D_\uparrow$. If we pose query $\mathcal{Q} : \pi_C(\sigma_{A=a_2} R)$ to $D_\uparrow$, we obtain $\mathcal{Q}(D_\uparrow) = \{\langle c_1, c_2, c_3\rangle\}$. Observe that $Poss_{\mathcal{Q}}(D_0) = \{\langle c_1, c_2, c_3\rangle\}$, and thus $\mathcal{Q}(D_\uparrow)$ provides an over-approximation for $Poss_{\mathcal{Q}}(D_0)$. It can be seen that an arbitrary (D_0, Σ)-clean instance, say D_2 for instance, may not provide a complete approximation to possible answer since $Poss_{\mathcal{Q}}(D_0) \not\sqsubseteq \mathcal{Q}(D_2) = \{\langle c_1, c_2\rangle\}$. ■

7. A CASE FOR SWOOSH'S ENTITY RESOLUTION

In [8], a generic conceptual framework for entity resolution is introduced. It considers a general match relation M, which is close to our similarity predicates $\approx$, and a general merge function, μ, which is close to our m functions. In this section we establish a connection between our MD framework and Swoosh. However, a full comparison is problematic, for several reasons, among them: (a) Swoosh works at the record (tuple) level, and we concentrate on the attribute level. (b) Swoosh does not use tuple identifiers and some tuples may be discarded, those that are dominated by others in the instance. The main problem is (a). However, to ease the comparison, we consider a particular (but still general enough) case of Swoosh, namely the combination of the *union case* with *merge domination*. In the following we embed this case of Swoosh into our MD framework, thus showing the power of the latter.

Although it is not explicitly said in [8], it is safe to say that the conceptual framework is applied to ground tuples of a single relational predicate, say R, which are called *records* there. In consequence, *Rec* denotes the set of ground tuples of the form $R(\bar{s})$. If the attributes of R are $A_1, \ldots, A_n$, then the component s_i of $\bar{s}$ belongs to an underlying domain Dom_{A_i}.

Relation M maps $Rec \times Rec$ into $\{true, false\}$. When two tuples are similar and have to be merged, M takes the value *true*. Moreover, μ is a partial function from $Rec \times Rec$ into *Rec*. It produces the merge of the two tuples into a single tuple, and is defined only when M takes the value *true*.

Now, the union case for Swoosh arises when the merge function μ produces the union of the records, defined as the component-wise union of attribute values. This latter union makes sense if the attribute values are sets of values from an even deeper data domain.

More precisely, for each of the n attributes A_i of R, we consider a possibly denumerable domain D_{A_i}. (Repeated attributes in R share the same domain, but it is conceptually simpler to assume that attributes are all different.) Each D_{A_i} has a similarity relation $\approx_{A_i}$, which is reflexive and symmetric. Now, for each attribute A_i of R, its domain becomes $Dom_{A_i} := \cup_{k\in\mathbb{N}} \mathcal{P}^k(D_{A_i})$, where $k > 0$ and $\mathcal{P}^k(D_{A_i})$ denotes the set of subsets of D_{A_i} of cardinality k. Thus, the elements of *Rec* are of the form $R(s_1, \ldots, s_n)$, with each s_i being a set that belongs to Dom_{A_i}. An initial instance D, before any entity resolution, will be a finite subset of *Rec*, and each attribute value in a record, say s_i for A_i, will be a singleton of the form $\{a_i\}$, with $a_i \in D_{A_i}$.

The $\approx_{A_i}$ relation on D_{A_i} induces a similarity relation $\approx_{\{A_i\}}$ on Dom_{A_i}, as follows: $s_1 \approx_{\{A_i\}} s_2$ holds iff there exist $a_1 \in s_1, a_2 \in s_2$ with $a_1 \approx_{A_i} a_2$. Each $\approx_{\{A_i\}}$ is reflexive and symmetric. ($s \approx_{\{A_i\}} s$, because there is $a \in s$ and $\approx_{A_i}$ is reflexive; and symmetry follows from the symmetry of $\approx_{A_i}$.) We also consider matching functions $m_{\{A_i\}} : Dom_{A_i} \times Dom_{A_i} \to Dom_{A_i}$ defined by $m_{\{A_i\}}(s_1, s_2) := s_1 \cup s_2$. The structures $\langle Dom_{A_i}, \approx_{\{A_i\}}, m_{\{A_i\}}\rangle$ have all the properties described in Sections 2 and 3.

PROPOSITION 9. Each matching function $m_{\{A_i\}}$ is total, idempotent, commutative and associative. It is also similarity preserving w.r.t. the $\approx_{\{A_i\}}$ similarity relation. ■

Now, based on [8], we are ready to define the "union match and merge case" for Swoosh. Consider two elements of *Rec*, say $r_1 = R(\bar{s}^1), r_2 = R(\bar{s}^2)$: (a) $M(r_1, r_2) := true$ iff for some i, $s_i^1 \approx_{\{A_i\}} s_i^2$. (b) When $M(r_1, r_2) := true$, $\mu(r_1, r_2) := R(m_{\{A_1\}}(s_1^1, s_1^2), \ldots, m_{\{A_n\}}(s_n^1, s_n^2))$.

Function M is reflexive and commutative, which follows from the reflexivity and symmetry of the $\approx_{\{A\}}$. From [8, Prop. 2.4] we obtain that the combination of M and μ has Swoosh's ICAR properties, namely:[3]

- I^s: Idempotency: $\forall r \in Rec, M(r, r)$ holds, and $\mu(r, r) = r$.
- C^s: Commutativity: $\forall r_1, r_2 \in Rec, M(r_1, r_2)$ iff $M(r_2, r_1)$. Also $M(r_1, r_2)$ implies $\mu(r_1, r_2) = \mu(r_2, r_1)$.
- A^s: Associativity: $\forall r_1, r_2, r_3 \in Rec$, if $\mu(r_1, \mu(r_2, r_3))$ and $\mu(\mu(r_1, r_2), r_3)$ exist, then they are equal.
- R^s: Representativity: $\forall r_1, r_2, r_3, r_4 \in Rec$, if $r_3 = \mu(r_1, r_2)$ and $M(r_1, r_4)$ holds, then $M(r_3, r_4)$ also holds.

Now, Swoosh framework with M and μ on Dom_A can be reconstructed by means of the following set Σ^S of MDs: For $1 \le i, j \le n$,

$$R[A_i] \approx_{\{A_i\}} R[A_i] \longrightarrow R[A_j] \rightleftharpoons R[A_j]. \qquad (6)$$

The RHS of (6) has to be applied, as expected, with the matching functions $m_{\{A_j\}}$. From Propositions 1 and 9, we obtain that there is a single (D, Σ^S)-clean instance D^m. Consistently with our MD framework, we will assume that records have tuple identifiers. Actually, in order to make

[3]We use the superscript s, for Swoosh, to distinguish them from the properties listed in Section 3.

more clear the comparison between the two frameworks, in this section and for the MD framework, we will use explicit tuple ids. They will be positioned in the first, extra attribute of each relation. When the MDs are applied, only the new version of a tuple is kept.

In the case of Swoosh, the application of μ generates a new, merged tuple, but the old ones may stay. However, Swoosh applies a pruning process based on an abstract domination partial order between records, $\preceq^S$. The framework concentrates mostly on the *merge domination relation* $\leq$, which is defined by:

$$r_1 \leq r_2 \ :\Longleftrightarrow \ M(r_1, r_2) = \mathit{true} \text{ and } \mu(r_1, r_2) = r_2. \quad (7)$$

The $I^sC^sA^sR^s$ properties make $\leq$ a partial order with some pleasant and expected monotonicity properties [8].

According to Section 3, we may consider each of the partial orders $\preceq_{\{A_i\}}$ on the Dom_{A_i}: $s \preceq_{\{A_i\}} s' :\Leftrightarrow m_{\{A_i\}}(s, s') = s'$. They induce a $\preceq$ relation on *Rec* (cf. Definition 2).

PROPOSITION 10. The general dominance relation $\preceq$ on *Rec* coincides with the merge domination relation $\leq$ obtained from M and μ. ■

Given a dirty instance D, it is a natural question to ask about the relationship between the clean instance D^m obtained under our approach, by enforcing the above MDs, and the *entity resolution* instance D^s obtained directly via Swoosh. The entity resolution D^s is defined in [8, Def. 2.3] through the conditions: 1. $D^s \subseteq \bar{D}$. 2. $\bar{D} \leq D^s$. 3. D^s is $\subseteq$-minimal for the two previous conditions. Here, the partial-order $\leq$ between instances is induced by the partial order $\leq$ between records as in Definition 2. Instance $\bar{D}$ is the *merge closure* of D, i.e., the $\subseteq$-minimal instance that includes D and is closed under M: $r_1, r_2 \in \bar{D}$ and $M(r_1, r_2) = \mathit{true} \ \Rightarrow \mu(r_1, r_2) \in \bar{D}$.

Notice that, in order to obtain D^m, tuple identifiers are introduced and kept, whereas under Swoosh, there are no tuple identifiers and new tuples are generated (via μ) and some are deleted (those $\leq$-dominated by other tuples). In consequence, the elements of D and D^m under the MD framework are of the form $R(t, s_1, \ldots, s_n)$, and those in D and D^s under Swoosh are the records r of the form $R(s_1, \ldots, s_n)$. Since t is a tuple identifier, for every $R(t, s_1, \ldots, s_n)$, $r(t)$ denotes the record $R(s_1, \ldots, s_n)$.

PROPOSITION 11. (a) For every r in D^s there is a tuple in D^m with tuple identifier t, such that $r(t) = r$.
(b) For every tuple $t \in D^m$, there is a record $r \in D^s$, such that $r(t) \leq r$. ■

8. CONCLUSIONS

The introduction of matching dependencies (MDs) in [17] has been a valuable addition to data quality and data cleaning research. They can be regarded as *data quality constraints* that are declarative in nature and are based on a precise model-theoretic semantics. They are bound to play an important role in database research and practice, together (and in combination) with classical integrity constraints.

In this work we have made several contributions to the semantics of matching dependencies. We have refined the original semantics introduced in [18], addressing some important open issues, but we have also introduced into the semantic framework the notion of matching function. This is an important ingredient in entity resolution since matching functions indicate how attribute values have to be merged or identified.

The matching functions, under certain natural assumptions, induce lattice-theoretic structures in the attributes' domains. This led us to introduce a partial order of domination between instances, and allowed us to compare them in terms of information content. The same domination order was then applied to sets of query answers. We also investigated the interaction of matching functions with similarity relations in the attribute domains.

On the basis of all these notions, we defined the class of clean instances for a given dirty instance. They are the intended and admissible instances that could be obtained after enforcing the matching dependencies. The clean instances were defined by means of a chase-like procedure that enforces the MDs, while not making unjustified changes on other attribute values. The chase procedure stepwise improves the information content as related to the domination order.

The notion of clean answer to a query posed to the dirty database was defined as a pair formed by a lower and an upper bound in terms of information content for the query answers. In this context we studied the notion of monotone query w.r.t. to the domination order and how to relax a query into a monotone one that provides more informative answer than the original one.

The domination-monotone relational query language introduced uses the lattice-theoretic structure of the domains, and is interesting in its own right. It certainly deserves further investigation, independently of MDs. It is an interesting open question to explore its connections with querying databases over partially ordered domains, with incomplete or partial information [31, 23, 28, 27], with query relaxation in general [26, 20], and with relational languages based on similarity relations [24].

We addressed some problems around the enforcement of a set of matching dependencies for purposes of data cleaning based on the original proposal of [17, 18], by explicitly making use of matching functions. We studied issues such as the existence and uniqueness of clean instances, the computational cost of computing them, and the complexity of computing clean answers. We identified cases where clean query answering is tractable, e.g., when there is a single clean instance. However, we established that this problem is intractable in general. We proposed polynomial time approximations.

Identifying other tractable cases and more efficient approaches to the intractable ones is part of ongoing research. We are currently investigating the use of logic programs with stable model semantics in the specification of clean instances and in clean query answering. This idea has been investigated in consistent query answering and has led to useful insights and implementations [4, 21, 7, 15, 13]. This route would avoid the explicit computation of the clean instances, and clean query answering could be done on top of the program. However, the lattice-theoretic structure of the domains and the domination order create a scenario that is substantially different from the one encountered in database repairs w.r.t. classical integrity constraints.

Acknowledgments. This work was supported by NSERC Strategic Network on Business Intelligence (BIN ADC01) and NSERC/IBM CRDPJ/371084-2008, which is gratefully

acknowledged.

9. REFERENCES

[1] Abiteboul, S., Kanellakis, P.C., Grahne, G. On the Representation and Querying of Sets of Possible Worlds. *Theoretical Computer Science*, 1991, 78(1):158–187.

[2] Afrati, F. and Kolaities, Ph. Answering Aggregate Queries in Data Exchange. In Proc. ACM PODS'08, 2008, pp. 129-138.

[3] Arenas, M., Bertossi, L. and Chomicki, J. Consistent Query Answers in Inconsistent Databases. In Proc. ACM PODS'99, 1999, pp. 68-79.

[4] Arenas, M., Bertossi, L. and Chomicki, J. Answer Sets for Consistent Query Answering in Inconsistent Databases. *Theory and Practice of Logic Programming*, 2003, 3(4-5):393-424.

[5] Arenas, M., Bertossi, L., Chomicki, J., He, X., Raghavan, V. and Spinrad, J. Scalar Aggregation in Inconsistent Databases. *Theoretical Computer Science*, 2003, 296(3):405-434.

[6] Bancilhon, F. and Khoshafian, S. A Calculus for Complex Objects. *Journal of Computer ans Systems Sciences*, 1989, 38(2):326-340.

[7] Barcelo, P., Bertossi, L. and Bravo, L. Characterizing and Computing Semantically Correct Answers from Databases with Annotated Logic and Answer Sets. In *Semantics in Databases*, Springer LNCS 2582, 2003, pp. 7-33.

[8] Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., Euijong Whang, S. and Widom, J. Swoosh: A Generic Approach to Entity Resolution. *VLDB Journal*, 2009, 18(1):255-276.

[9] Bertossi, L. Consistent Query Answering in Databases. *ACM Sigmod Record*, 2006, 35(2):68-76.

[10] Bertossi, L., Kolahi, S. and Lakshmanan, L. Data Cleaning and Query Answering with Matching Dependencies and Matching Functions (full version). `http://people.scs.carleton.ca/~bertossi/papers/matchingDC-full.pdf`, 2010.

[11] Bleiholder, J. and Naumann, F. Data Fusion. *ACM Computing Surveys*, 2008, 41(1).

[12] Buneman, P., Jung, A. and Ohori, A. Using Powerdomains to Generalize Relational Databases. *Theoretical Computer Science*, 1991, 91(1):23-55.

[13] Caniupan, M. and Bertossi, L. The Consistency Extractor System: Answer Set Programs for Consistent Query Answering in Databases. *Data & Knowledge Engineering*, 2010, 69(6):545-572.

[14] Chomicki, J. Consistent Query Answering: Five Easy Pieces. Proc. ICDT 2007, Springer LNCS 4353, pp. 1-17.

[15] Eiter, T., Fink, M., Greco, G. and Lembo, D. Repair Localization for Query Answering from Inconsistent Databases. *ACM Transactions on Database Systems*, 2008, 33(2).

[16] Elmagarmid, A., Ipeirotis, P. and Verykios, V. Duplicate Record Detection: A Survey. *IEEE Transactions in Knowledge and Data Engineering*, 2007, 19(1):1-16.

[17] Fan, W. Dependencies Revisited for Improving Data Quality. In Proc. ACM PODS'08, 2008, 159-170.

[18] Fan, W., Jia, X., Li, J. and Ma, S. Reasoning about Record Matching Rules. PVLDB, 2009, 2(1):407-418.

[19] Fuxman, A. and Miller, R. First-Order Query Rewriting for Inconsistent Databases. *J. Computer and System Sciences.*, 2007, 73(4):610-635.

[20] Gaasterland, T., Godfrey, P. and Minker, J. Relaxation as a Platform for Cooperative Answering. *J. Intelligent Information Systems*, 1992, 1(3/4):293-321.

[21] Greco, G., Greco, S. and Zumpano, E. A Logical Framework for Querying and Repairing Inconsistent Databases. *IEEE Transactions on Knowledge and Data Engineering*, 2003, 15(6):1389-1408.

[22] Gunter, C.A. and Scott, D.S. Semantic Domains. Chapter 12 in *Handbook of Theoretical Computer Science*, Vol. B. Elsevier, 1990.

[23] Imielinski, T. and Lipski Jr., W. Incomplete Information in Relational Databases. *Journal of the ACM*, 1984, 31(4):761-791.

[24] Jagadish, H., Mendelzon, A., and Milo, T. Similarity-Based Queries. Proc. ACM PODS'95, 2005, pp. 36-45.

[25] Kifer, M. and Lausen, G. F-Logic: A Higher-Order language for Reasoning about Objects, Inheritance, and Scheme. In Proc. SIGMOD'89, 1989, pp. 134-146.

[26] Koudas, N., Li, Ch., Tung, A. and Vernica. R. Relaxing Join and Selection Queries. In Proc. VLDB'06, 2006, pp. 199-210.

[27] Levene, M. and Loizou, G. Database Design of Incomplete Relations. *ACM Transactions on Database Systems*, 1999, 24:35-68.

[28] Libkin, L. A Semantics-Based Approach to Design of Query Languages for Partial Information. In *Semantics in Databases*, 1998, Springer LNCS 1358, pp. 170-208.

[29] Libkin, L. Data Exchange and Incomplete Information. In Proc. ACM PODS'06, 2006, pp. 60-69.

[30] Lipski Jr., W. On Semantic Issues Connected with Incomplete Information Databases. *ACM Transactions on Database Systems*, 1979, 4(3):262-296.

[31] Ng, W., Levene, M. and Fenner, T. On the Expressive Power of the Relational Algebra with Partially Ordered Domains. *International Journal of Computer Mathematics*, 2000, 71:53-62.